Inferix: Decentralized GPU Network with Proof of Rendering

Inferix Labs 2024, version 1.0

***Abstract*—This paper introduces Proof-of-Rendering (PoR) and its application in building Inferix's decentralized GPU network. Addressing DePIN Verification, the key challenge facing developers of decentralized physical infrastructure networks (DePIN) recently, Inferix has developed the Active Noise Generation (ANG), Active Noise Verification (ANV), and PoR algorithms. These algorithms are combined with a software layer that includes middleware and client SDK, facilitating connections between 3D creative data systems and the decentralized GPU infrastructure. This creates a unique Decentralized GPU Network for Visual Computing and AI Inference.**

***Index Terms*—Proof-of-Rendering, PoR, Active Noise Generation, Active Noise Verification, 3D/VR Rendering, Federated Learning, Inference, Burn-Mint-Work, IBME, Decentralization, GPU Compute.**

I. Introduction

Inferix is a DePIN network of GPUs for visual computing and AI, it is built to bridge the needs of users and hardware owners. Its solution meets real-world problems across a range of industries, not only for the AI field but also for high-quality graphics rendering. Users (e.g. 3D graphics artists, game developers, enterprises) who need GPU computing power for rendering high-quality graphics can use the Inferix system to continuously access these precious resources with faster processing time and more efficient spending. Owners of GPUs can share idle resources to the Inferix network and earn long-term passive income while simultaneously balancing their main jobs or leisure activities.

At high-level, Inferix network is naturally a dynamic system where demands of digital content creators and supplies of GPU owners are created continuously over time. Users are concerned with the security and privacy of the system, with the facility of accessing computing resources, as well as with the price that they have to pay for their demands.

This section first describes the high-level flow of a decentralized rendering network. Next, we describe one of the main challenges that we have to deal with, that is the authenticity of rendering. Section II presents the main idea of the proposed solution then introduces a mathematical model for the Active Noise Generation and Verification algorithm. Section III describes an implementation for the algorithm and its integration into the existing layers of the Inferix network. Section IV presents the main components in the system architecture of the Inferix decentralized GPU network. In section V, we discuss how to use this network infrastructure for the AI training and inference, then Inferix is actually a GPU network for visual computing and federated AI. In section VI, we present the token economy model of Inferix with a novel algorithm called Burn-Mint-Work for the token issuance problem. Finally, section VII is reserved for ongoing developments in improving the robustness, performance and availability of the network.

A. Rendering network using crowdsourced GPU

The graphics rendering service consists in a network of decentralized machines called *nodes* which are of 3 kinds: *manager*, *worker* and *verifier*. The *managers* are dedicated machines of Inferix while *verifiers* and *workers* are machines joined by GPU owners. The number of *workers* is normally much larger than the number of *managers* and *verifiers*. A typical rendering session contains several steps shown in fig. 1:

1) A user creates a rendering job request using the Inferix's plugin for client, this job uploads user's scene data to some *manager*.
2) The rendering task controller of the *manager* receives the rendering job request, then
   - splits it into multiple rendering tasks, each consists of the scene data and several parameters: range of frames to be rendered, output format, etc.
   - generates corresponding verification keys and sends these keys to the verifying task controller.

   The rendering tasks will be assigned for *workers* and the verification keys will be sent to the verifying task controller.
3) Receiving a rendering task, a *worker* renders the included scene using the parameters given by the task. When it finishes, it saves the rendered frames to a decentralized storage, then notifies the *manager* by a message containing a unique URL to the result.
4) The verifying task controller of the notified *manager* receives the notification then creates a verification task, this task will be assigned to a *verifier*.
5) Receiving a verification task, a *verifier*
   - checks the authenticity of the corresponding rendered frames, then
   - notifies the *manager* about the verification result.
6) If the rendered frames pass the verification then the *manager* notifies the user by a message containing the URL to the rendered frames. Otherwise, these frames are rejected.
7) The user downloads the frames from the storage and manually confirms whether they meet the expectation, if they do not then the user sends a bad result claim to the *manager*.

The *managers* synchronize a database of rendering and verification tasks. That makes the rendering service being both

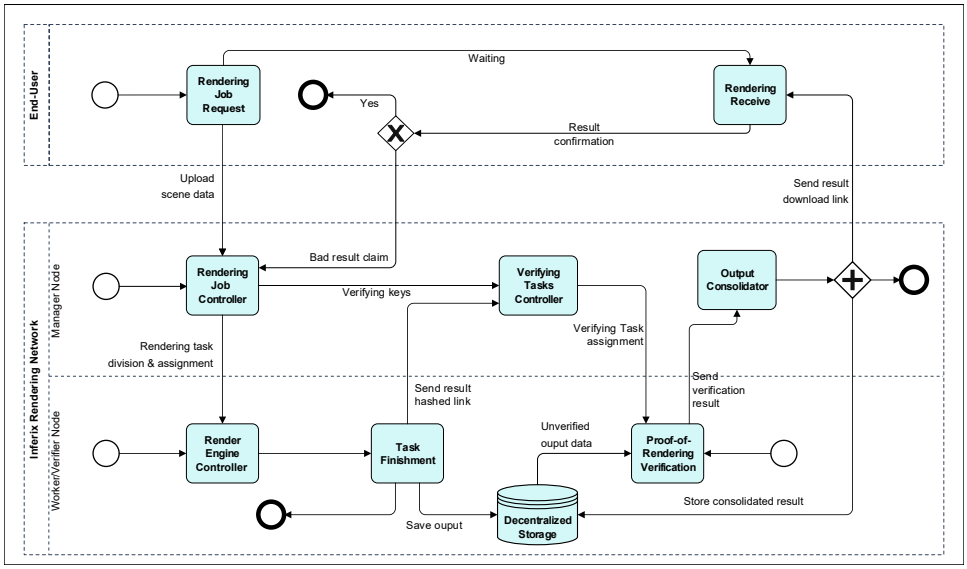


Figure 1: Rendering flow

logically and physically decentralized: a graphics scene can be simultaneously rendered by different *workers* and later checked by different *verifiers*, the machines of *workers* and *verifiers* can be also located at different geographical locations.

B. Rendering verification problem

A user submits some graphics work to a *manager* (cf. fig. 1), this work consists of several scenes; each contains information about graphical objects, the camera, light sources and materials. The photorealistic rendering consists of sophisticated computation processes that calculate light properties at surfaces of all visible objects, resulting in 3D rendered images of the scene [1].

One of the most important problems that Inferix has to solve is to verify the *authenticity* of rendered results [2], [3], [4]. That means how to ensure that once a user submits a valid scene, then after waiting for an amount of time, the user will receive authentically rendered images. The authenticity can be defined informally as if the result received from the rendering network and the result received when the scene is genuinely rendered by a rendering software are human perceptually indistinguishable.

The *workers* joined the rendering network are mostly workstations of GPU owners who want to make profit from their unused computational resources. Respecting the privacy of GPU owners and their resources, besides lightweight open source software installed to manage the communication with the network, there is completely no control on *workers*.

Consequently, there is no constraint to oblige *workers* to render the graphics scene correctly. Indeed, a malicious *worker* may receive a rendering task, but does nothing then uses some forged images as results. Without rendering the scene, the *managers* and users know only superficial features of what the rendered images look like. Obviously, the *managers* and users have no interest in rendering the scene themselves since if they can do that, there is no need to rely on *workers*. Moreover, we cannot deploy any surveillance mechanism on the machines of *workers* due to privacy reasons. Even if we try to do that, this is only a matter of time before a *worker* reverse engineers the mechanism and eventually bypasses it. The situation doesn't seem to favor us: checking the authenticity of something while only having a little knowledge about it, otherwise the attacker has complete information.

Naturally, a public-key cryptography approach is using a scheme of *fully homomorphic encryption* (FHE) [5]. The scene is encrypted first by a private key before sending it to *workers*. Given the corresponding public key, the homomorphic encryption software performs the graphics rendering on the encrypted scene without needing to decrypt it. Finally, the encrypted rendered results are returned and decrypted at the user's side using the private key. The advantage of FHE is that the *workers*, even being able to modify the FHE software on their side, cannot interfere with the FHE rendering processes or forge the rendering results without being detected. Unfortunately, this approach is impractical since all state-of-the-art implementations will make the performance of the homomorphic encryption rendering become unacceptable [6].

II. High-level description

To handle this problem, we follow the approach of digital watermarking [7], [8] and propose a scheme called *Active Noise Generation and Verification* (ANGV) which is a variant of *proof of ownership* [2], [3]. Our scheme has several favorable properties:

- Efficiency: noise generation and verification requires much lower computational resources compared with the graphics rendering, the total performance of the system is not affected.
- Fidelity: the scheme needs to modify the initial scene so the rendered output will be distorted, but the distortion is human perception subthreshold.

- Robustness: the embedded noises are robust under rendering enhancements and post-processing operations (e.g. denoising, anti-aliasing).
- Effectiveness: there is no need to use a special rendering software as in the case of FHE.
- Security: without knowing the verification key, attackers need the same computational cost with the rendering to bypass the authenticity verification.

In current digital watermarking schemes for authentication and ownership verification [2], [3], [4], [9], invisible watermarks will be embedded into the digital content needed to be protected. The detector (or verifier) tries to extract the watermark from a tested content, then compares the extracted watermark with the original embedded one, if the comparison is passed then the content is authenticated.

However, in the context of Inferix's rendering network, the *manager* has access to the image only after the graphics scene has been rendered by *workers*. It is nonsense to embed watermark into the image at this point since the watermarking cannot help to detect any malicious manipulations which may happen before that, i.e. in the rendering process. Our approach is to embed watermarks into the graphics scene submitted by users before sending it to *workers*. The *Active Noise Generation and Verification* scheme consists of two algorithms as described below.

A. Noise generation

In practice, a scene may contain multiple frames, each task of this scene contains some range of frames to be rendered, consequently each worker may render only a subset of these frames. For the simplification purpose, we assume in this section that a scene has only one frame, so the output image is determined uniquely by the scene.

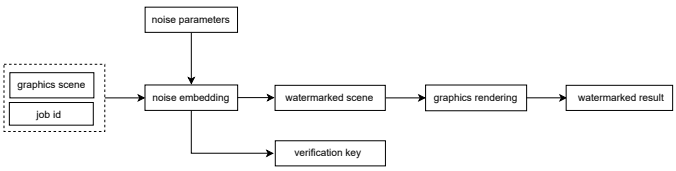


Figure 2: Noise generation

Let $\mathcal{R}$ denote the rendering process, for each input scene G, the result of the rendering is an image:

$$I = \mathcal{R}(G) \tag{1}$$

It is important to note that I is actually never computed, neither by the *manager* in the noise embedding (see also the discussion about sampling frames in section III-C) nor by *workers* in the frame rendering. The equation above represents only equality.

Similar with invisible watermark schemes in the literature [2], [3], [9], a noise W consists in a random vector of atomic watermarks:

$$W \triangleq (w_1, \ldots, w_n) \tag{2}$$

where w_i $(1 \leq i \leq n)$ is independently chosen from some normal probability distribution $\mathcal{N}(\mu, \sigma^2)$. Furthermore, w_i has a special structure depending on where it is introduced in the scene G. The number n of atomic watermark signals is chosen around an experimental trade-off between human perception threshold about the image distortion and the false positive ratio of the noise verification.

Using a uniformly generated task identification number $J_{\mathtt{id}}$, we calculate a verification key which is a vector of the same size as the noise vector W:

$$K_{\mathtt{verif}}(S, W, J_{\mathtt{id}}) \triangleq (k_1, \ldots, k_n) \tag{3}$$

that will be used later for the noise verification.

We have discussed that embedding watermarks into I cannot help the authentication, then the noise W is not embedded into the image I but into the scene G (c.f. fig. 2). Let $\mathcal{E}$ denote the embedding function, we now create a watermarked scene:

$$\hat{G} = \mathcal{E}(G, W) \tag{4}$$

Finally $\hat{G}$ is sent to *workers* for rendering, that results in a rendered image:

$$\hat{I} = \mathcal{R}(\hat{G}) \tag{5}$$

If got accepted, namely $\hat{I}$ passes the noise verification which will be presented hereafter, this is the image sent back to the user (recall that I in eq. (1) is not computed). The encoding function $\mathcal{E}$ and the noise W are designed so that the distortion of $\hat{I}$ against I is imperceptible [10], [11], then $\hat{I}$ can be authentically used as a result of the graphics rendering.

B. Noise verification

Different from proof of ownership schemes [2], [3], the verification of watermark requires a key. Given an image J and a verification key $K_{\mathtt{repr}}$ (c.f. eq. (3)), we first try to recover a watermark $\hat{W}$ from J using a decoding function $\mathcal{D}$:

$$\hat{W} = \mathcal{D}(J, K_{\mathtt{repr}}) \tag{6}$$

Next $\hat{W}$ is compared against W, if the deviation is above some threshold T:

$$\|\hat{W} - W\| \geq T \tag{7}$$

then J will be accepted otherwise rejected.

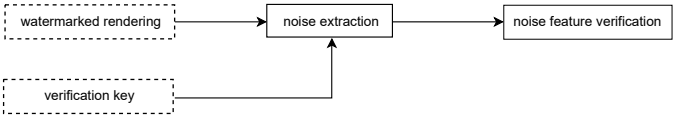


Figure 3: Noise verification

C. Threat model

Given rendering tasks each contains basically a watermarked scene $\hat{G}$ and some range of frames required to be rendered, the goal of an attacker, namely a malicious *worker* (or in general a group of maliciously colluding *workers* [12]), is to generate rendered frames that pass the noise verification, with computational costs significantly lower than doing render this range by some conventional rendering software.

By Kerckhoff's principle, it is essential that the attackers know the noise generation and verification algorithms, working

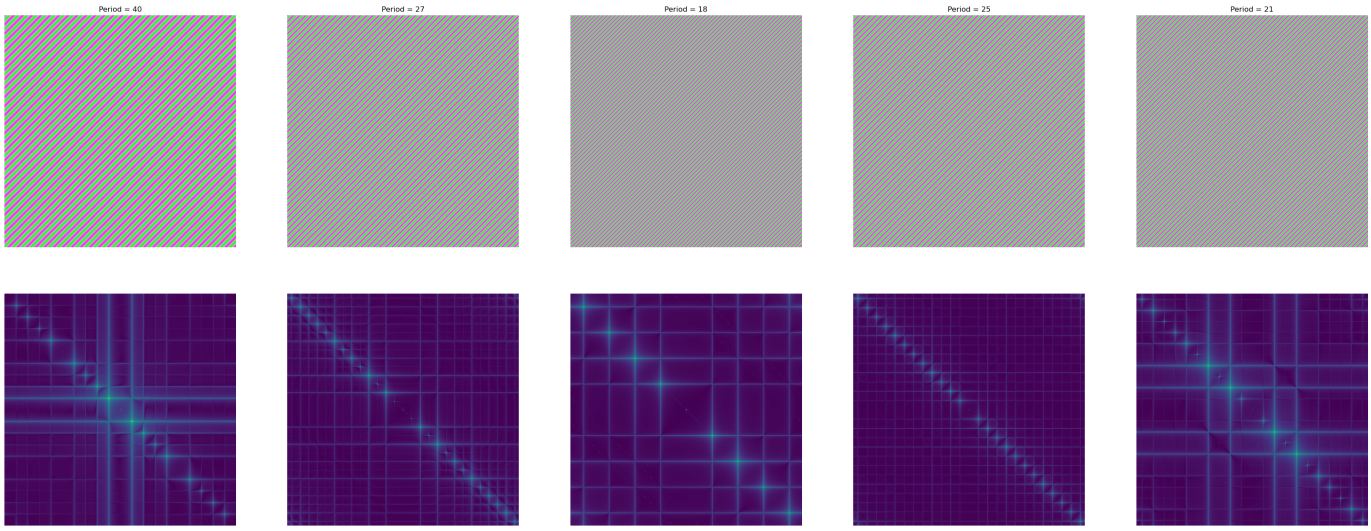


Figure 4: A random vector of 5 atomic watermarks (period $\sim \mathcal{N}(25, 5)$)

parameters including trade-offs. But the task identification numbers and the corresponding verification keys are kept secret. Furthermore, we require a strong assumption that the attackers cannot detect the existence of watermarks in scenes. That means attackers can analyze and even modify different watermarked scenes $\hat{G}$(s) but they cannot distinguish objects of the noise wrapping vector Ω (discussed in detail in section III-B) embedded in $\hat{G}$(s) from original graphical objects of G. Otherwise, we assume no constraint on the communication capability of colluding attackers.

Not surprisingly, the security of ANGV can be modeled as the problem of sending steganographic messages over a public communication channel with passive adversaries [13], [14]. Indeed, let us consider some graphics scene, by repetitively receiving rendering tasks for this scene and sending results (both genuinely rendered and intentionally forged), an attacker (or set of colluding attackers) knows a set of accepted and rejected images. The attacker analyzes these tested images to estimate probability distributions $P_{\mathcal{S}}$ and P_C for respectively images that would pass the noise verification and images that would be rendering results of the scene. We use the traditional notations of the steganography literature: C for cover-work and S for stego-work [15]. It may be worth noting that $P_{\mathcal{S}}$ and P_C represent partial knowledge of the attacker obtained by analyzing the set of tested images: the larger this set, the more precise estimation for $P_{\mathcal{S}}$ and P_C.

The information-theoretic security of ANGV is quantified by the Kullback–Leibler divergence (i.e. relative entropy) $D(P_C \parallel P_{\mathcal{S}})$ of P_C from $P_{\mathcal{S}}$. Concretely, ANGV is called ϵ-secure if

$$\lim_{n\to\infty} D(P_C \parallel P_{\mathcal{S}}) \leq \epsilon \tag{8}$$

where n is the number of tested images. In particular, $\epsilon = 0$ if and only if $P_C = P_{\mathcal{S}}$, or the attacker cannot distinguish watermarked images from genuinely rendered ones, in this case we have perfect security.

III. Implementation

Section II presents a high-level description of the Active Noise Generation and Verification algorithm. In this section, we discuss in detail the current implementation approaches and proposed trade-off values.

A. Structure of noise

As introduced in eq. (2), a noise W is a random vector $(w_i)_{1\leq i\leq n}$ where each element is independently chosen from a normal distribution. This atomic watermark is constructed as a rectangular image of periodic patterns as follows:

- let fix some values M, N for the width and the height of the rectangle, and
- let $\mathcal{X}_i, \mathcal{Y}_i$ be independent and identically distributed normal random variables:

$$\mathcal{X}_i \sim \mathcal{Y}_i \sim \mathcal{N}\left(\mu, \sigma^2\right) \tag{9}$$

for some μ and σ, then take X_i, Y_i be respectively some samples of $\mathcal{X}_i, \mathcal{Y}_i$.

The complex atomic signal w_i is defined by:

$$w_i(x, y) = Ae^{2i\pi\left(\frac{x}{X_i}+\frac{y}{Y_i}\right)} \quad (0 \leq x < M, 0 \leq y < N) \tag{10}$$

for some amplitude A. We observe that

$$w_i(x, y) = w_i(x + X_i, y) = w_i(x, y + Y_i) \ \forall x, y$$

then X_i, Y_i are actually the horizontal and the vertical periods.

Remark. *$\mathcal{X}_i$ and $\mathcal{Y}_i$ are elements of a set $\{\mathcal{X}_i, \mathcal{Y}_i \mid 1 \leq i \leq n\}$ of independent and identically distributed normal random variables $\mathcal{N}(\mu, \delta^2)$. The parameters μ and σ are chosen by analyzing the input scene that is discussed in section III-C.*

Proposition 1 (Fourier transform of complex atomic signals)**.**

$$F_i(u, v) = \frac{A}{M \times N} \frac{\left(1 - e^{2i\pi\frac{M}{X_i}}\right)\left(1 - e^{2i\pi\frac{N}{Y_i}}\right)}{\left(1 - e^{2i\pi\left(\frac{1}{X_i}+\frac{u}{M}\right)}\right)\left(1 - e^{2i\pi\left(\frac{1}{Y_i}+\frac{v}{N}\right)}\right)}$$

Proof. Direct calculation (for details, see appendix A-A). □

The structure of noise given in eq. (10) has two folds: we empirically find that this form of signal makes the wrapping graphical objects (discussed in section III-B) persistent in the rendering of graphics scenes. Furthermore, the distortion raised by any atomic watermark is easily controlled thanks to the simple form of the signal amplitude given in proposition 1.

The spectrums of atomic signals play a crucial role in the noise verification since they help to distinguish embedded noises from the original image signals. They are also completely determined by the periods X_i, Y_i given fixed M, N since the discrete Fourier transform in proposition 1. In turn, these periods statistically rely on the expectation μ by eq. (9), we will discuss how to choose this value in section III-C.

The length n of the noise vector is one of the principal factors which decides the robustness of noise: the higher the value n, the lower the false positive of noise verification. But this size influences the quality of the rendered image: the lower value n, the higher fidelity of the rendered images. Consequently, the value n is a trade-off between the robustness of the embedded noise and the fidelity of the rendered image, it is empirically chosen to be about 8 to 15.

The fig. 4 shows a noise as a vector of 5 atomic watermarks and the Fourier transforms showing the corresponding frequency characteristics. The vital frequencies of energies are clearly shown in the spectrums. For illustration purpose, we take $X_i = Y_i \sim \mathcal{N}(25, 5)$ $(1 \le i \le 5)$, and $M = N = 512$.

B. Noise insertion

Given a scene G, the random vector $W = (w_i)_{1 \le i \le n}$ is embedded into G by first wrapping each atomic w_i by a graphical object: let denote it ω_i, then we obtain a vector of objects $\Omega = (\omega_i)_{1 \le i \le n}$. Next we insert Ω into G so that every ω_i contributes to the rendered image, namely they distort this image. The distortion is kept to be lower than the human perception of light [10], [11].

1) Geometric constraints: By the nature of physically based rendering [1], an object (or any part of it) in the scene will not be visible if and only if there is no visible light (or in general the light is out of the capability of the sensor) scattered from the surface of the object to the digital camera object. This may be caused by several reasons: the object is not located in the frustum of the camera, is hidden by other objects, or the object is made of some transparent material. Furthermore, the atomic signals w_i $(1 \le i \le n)$ should not interfere themselves since this makes the noise verification to be unnecessarily complicated. Consequently, we require the noise embedding to satisfy first the following constraints:

- there are no collisions between ω_i $(1 \le i \le n)$,
- Ω is completely located in the camera frustum,
- no ω_i is hidden by another object (even partially), including both $\omega_j \in \Omega$, $j \ne i$ and objects of the scene.

As mentioned in section III-A, the characteristics of atomic noises in frequency domain are crucial for the robustness of noise verification: we need to restore a certain amount of information about these characteristics from very small distortions made by the objects $\omega_i \in \Omega$ on rendered images. Because of the unavoidable requirement about the fidelity of images, we have to keep these distortions local, concretely these distortions must be well-placed on regions whose locations can be pre-calculated. A practical approach is to constrain the distortion made by ω_i to be of the same shape as the atomic noise w_i. Geometrically, each ω_i has a *rotation vector* which characterizes the direction of the object in the global coordinate system (i.e. world space [1]) containing all objects of the scene G. To keep the rectangular shape of the distortion of ω_i, we require that:

- the rotation vector of ω_i is equal with the rotation vector of the digital camera of G for all $\omega_i \in \Omega$.

2) Distortion region: Under constraints about position and direction of noise objects, the imprint of ω_i on the rendered image is a rectangular region denoted by:

$$k_i \triangleq \left(x_i^{\mathtt{ul}}, y_i^{\mathtt{ul}}, x_i^{\mathtt{lr}}, y_i^{\mathtt{lr}}\right) \tag{11}$$

where $\left(x_i^{\mathtt{ul}}, y_i^{\mathtt{ul}}\right)$ and $\left(x_i^{\mathtt{lr}}, y_i^{\mathtt{lr}}\right)$ are respectively the upper left and lower right positions in the image coordinate system. It is important to note that k_i for all $1 \le i \le n$ can be computed without rendering the scene G.

For the size of distortion regions, similar with the length of the noise random vector, there is a compromise between the robustness of the embedded noise and the fidelity of the rendered frame. The larger the distortion k_i, the higher information of w_i can be restored then the higher robustness of the noise verification; but the lower the distortion k_i, the higher fidelity of the image. Empirically, we use the bounds $4 \le x_i^{\mathtt{lr}} - x_i^{\mathtt{ul}},\ y_i^{\mathtt{lr}} - y_i^{\mathtt{ul}} \le 7$ for all $1 \le k \le n$.

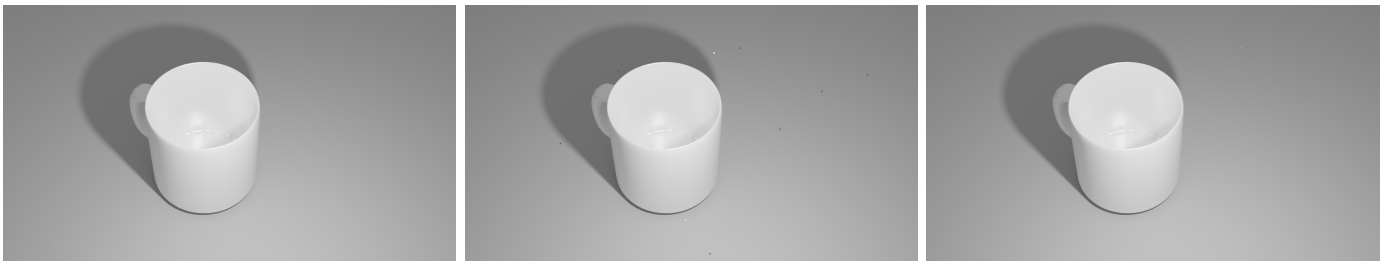
The fig. 5 shows some distortion results of rendering watermarked scenes. From two original scenes, noise vectors of length 12 with different distortion sizes are embedded, then different watermarked scenes are generated. When rendering the scenes containing noises whose distortion sizes are 7 or 8, the distortions are visible under the form of small rectangles dispersed in the rendered images. In contrast, when the sizes are 4 or 5, the distortions are imperceptible.

Remark. *While the atomic watermarks are quite large, the distortions made by them on rendered images are constrained relatively small. The fig. 4 shows atomic watermarks of size* 512×512 *which are used for watermarking scenes shown in fig. 5, their imprints are about* 4×4. *The sizes of the rendered images are much larger:* 1080×1080 *and* 1920×1080.

C. Adaptive noise spreading

As discussed above, each atomic $w_i \in W$ has its distortion contribution at the spatial region k_i on the rendered image. Since the rendering process contains multiple options to improve the quality of the output (noise reduction, anti-aliasing, etc.), it is severe if the region k_i(s) fall into perceptually insignificant regions [7] of the image because the deliberate distortions raised by w_i(s) would be eliminated by the rendering enhancement. Hence, distortions are preferred to be placed

(a) Source: Coca Cola bottle (b) Distortion size = 8 (c) Distortion size = 5



(d) Source: Glass Tea mug (e) Distortion size = 7 (f) Distortion size = 4

Figure 5: Rendered watermarked graphics scenes (random vector length = 12)

in human perceptually significant regions. However, to keep the compromise between the fidelity of the rendered frames and the robustness of noise verification, the strength of the distortion of each w_i must be tuned so that its deviation from locally enclosed regions is within a predetermined bound.

We handle this problem using the adaptive noise spreading [16], [17], [18]. Research on the human visual perception agrees that the important information of images is located at high energy and low frequency spectral regions [10]. Then before embedding the object vector Ω, we render the scene G at some low settings to get a sampling instance of the image. Next, we proceed both the spatial and spectral analysis on this instance to get perceptually significant spatial regions, called preferred regions. The atomic noises w_i(s) will be placed in these regions. Simultaneously, we tune the expectation value μ used in generating atomic noises so that the energy (statistically given in proposition 2) of high frequencies of the noises are sufficiently higher than the threshold used in the noise verification (c.f. section III-E).

Proposition 2 (Convergence of energies)**.** *Let* $\{\mathcal{X}_i, \mathcal{Y}_i \mid i \in \mathbb{N}\}$ *be a set of independent and identically distributed normal random variables* $\mathcal{X}_i \sim \mathcal{Y}_i \sim \mathcal{N}(\mu, \sigma^2)$*. Let* ${}^t\begin{bmatrix} X_i & Y_i \end{bmatrix}$ *be a sample of the random vector* ${}^t\begin{bmatrix} \mathcal{X}_i & \mathcal{Y}_i \end{bmatrix}$ *and* w_i *be the signal* $(x, y) \mapsto Ae^{2i\pi\left(\frac{x}{X_i}+\frac{y}{Y_i}\right)}$ *for any* $i \in \mathbb{N}$*. Then the average of discrete Fourier transforms* $\overline{F}_n = \frac{1}{n}\sum_{i=1}^{n} F_i$ *converges:*

$$\overline{F}_n(u,v) \xrightarrow[n\to\infty]{a.s} \frac{A}{M \times N} \frac{\left(1 - e^{2i\pi\frac{M}{\mu}}\right)\left(1 - e^{2i\pi\frac{N}{\mu}}\right)}{\left(1 - e^{2i\pi\left(\frac{1}{\mu}+\frac{u}{M}\right)}\right)\left(1 - e^{2i\pi\left(\frac{1}{\mu}+\frac{v}{N}\right)}\right)}$$

for all $0 \le u < M,\ 0 \le v < N$.

Proof. Direct application of the continuous mapping theorem. For details, see appendix A-B. □

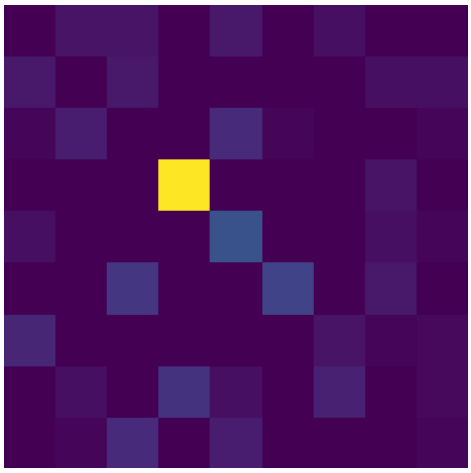
It is worth noting that the rendering work of G generally contains multiple tasks, each requires to render multiple image frames. But the number of reference instances used for spectral analysis is much smaller, practically less than 1% the total number of rendered frames. While keeping the robustness of ANGV, we can adjust this ratio be even smaller by increasing the size n of the noise vector W.

D. Verification key generation

The constraints and trade-offs discussed in sections III-B and III-C are to ensure the *fidelity* of rendered results and the *robustness* of the verification, but they do not concern the security. Indeed, any attacker knowing the algorithm and parameters including trade-offs, can straightforwardly generate (without rendering the graphics scene) forged images with the same spectral characteristics, finally bypasses the verification. The security is supported using verification keys.

	0	1	2	3	4	5	6	7	8	9
0	168 166 144	170 168 146	170 168 147	166 164 143	169 166 145	167 165 143	169 166 144	167 165 144	165 163 142	
1	170 168 146	168 166 145	170 168 146	167 165 144	167 165 144	168 165 144	167 165 144	168 166 145	168 166 144	
2	169 167 145	171 168 146	170 168 146	168 168 146	168 170 145	168 166 144	166 164 143	167 165 143	168 166 145	
3	167 164 143	168 166 145	167 172 149	198 186 164	165 169 148	167 165 144	167 165 144	169 167 145	169 166 145	
4	168 165 144	166 164 142	164 167 143	183 169 155	173 174 147	169 167 145	167 165 144	170 167 146	170 166 145	
5	166 164 143	166 164 143	170 168 146	166 164 143	169 166 145	172 169 148	166 164 143	169 167 146	168 165 144	
6	169 167 146	167 165 144	168 166 144	169 166 144	168 166 145	167 165 144	169 167 145	169 167 145	169 167 146	
7	166 164 143	169 166 145	168 166 144	171 169 147	170 168 146	168 166 145	170 168 146	169 165 144	170 167 146	
8	167 165 144	169 167 145	171 169 147	167 165 144	170 168 146	167 165 144	168 166 144	168 166 145	169 167 145	
9										

(a) Enveloping region (distortion region at the center)



(b) After applying the Laplacian filter

Figure 6: Noise verification using Laplacian filter

Each rendering task has a secret key, in current implementation, this key is also the task identification number $J_{\mathtt{id}}$. When embedding the noise vector W into the scene G, this number is used to compute distortion regions k_i for all $1 \le i \le n$, the vector $(k_i)_{1\le i\le n}$ is called verification key. The computation is modeled as a function (c.f. eq. (3)):

$$K_{\mathtt{verif}}\colon\ (S, W, J_{\mathtt{id}}) \mapsto (k_i)_{1\le i\le n}$$

In the operation of the rendering network, the leak of used verification keys is unavoidable. For instance, a *worker* may register itself to become a *verifier* node; when got accepted, it will be assigned verification tasks containing verification keys, then will be able to collect used keys. Even worse, colluding *workers* may exchange collected keys so that each of them will possess a much larger collection [12]. Another possibility is the malicious *workers* may get verification keys from some compromised *verifiers*. Hence, the $K_{\mathtt{verif}}$ must be designed so that the knowledge about used keys does not leak any information about the next generated keys. The following proposition is necessary for the security of ANGV.

Proposition 3. *$K_{\mathtt{verif}}$ is a cryptographic hash function.*

E. Noise verification

Given a tested image J and a verification key $K = (k_i)_{1\le i\le n}$, the goal of noise verification is to recover and check the trails of noises in J at all regions k_i. For each k_i, we pick an atomic enveloping region v_i determined by:

$$v_i \triangleq \left(x_i^{\mathtt{ul}} - \delta_i^x, y_i^{\mathtt{ul}} - \delta_i^y, x_i^{\mathtt{lr}} + \delta_i^x, y_i^{\mathtt{lr}} + \delta_i^y\right) \tag{12}$$

where δ_i^x and δ_i^y are the width and the height of k_i:

$$\delta_i^x = x_i^{\mathtt{lr}} - x_i^{\mathtt{ul}} + 1 \qquad \delta_i^y = y_i^{\mathtt{lr}} - y_i^{\mathtt{ul}} + 1 \tag{13}$$

Since any enveloping region is so small that spectral analysis cannot give reliable results, hence to filter the distortions of noises (i.e. the trails of high energy) we compare gradients of the region and the contained distortion region; one way to do that is using the Laplacian filter. Let ∇^2 denote the Laplacian operator, calculate the mean of each enveloping region v_i:

$$\overline{v}_i = \frac{1}{|v_i|} \sum_{(x,y)\in v_i} \left(\nabla^2 v_i\right)(x, y) \tag{14}$$

and the mean of corresponding distortion region:

$$\overline{k}_i = \frac{1}{|k_i|} \sum_{(x,y)\in k_i} \left(\nabla^2 v_i\right)(x, y) \tag{15}$$

where $|v_i|$ and $|k_i|$ are respectively the area of v_i and of k_i. Then compare the deviation (c.f. eqs. (6) and (7)):

$$e_i \triangleq |\overline{v}_i - \overline{k}_i| \tag{16}$$

with some energy threshold. Using the noise tuning discussed in section III-C, we experimentally accept the existence of the atomic watermarked w_i when $e_i \ge 5$.

If there is a distortion region where the deviation e_i is lower than the threshold then the image J is immediately rejected, otherwise J is accepted.

Remark. *From the construction of enveloping regions from distortion regions, the areas can be simply calculated by $|k_i| = \delta_i^x \times \delta_i^x$ and $|v_i| = 9 \times |k_i|$.*

The fig. 6a shows an enveloping region of size 9×9, its distortion region is of size 3×3 located at the center, numbers at each pixel are the RGB color values. The fig. 6b shows the enveloping region after applying the Laplacian convolution.

F. Threat analysis

The threat analysis of ANGV bases on the security model mentioned in section II-C. It requires careful and sophisticated settings (it may be worth noting that the rendering problem itself is undecidable in general [19]) then we refer the details to a technical report. We present only some basic results that can rapidly prove from current settings.

1) Attacks on verification keys: As discussed in section III-D, colluding attackers may know a set of used keys, then use these keys to predict the next keys (this is called *random number generator attack* [20]). Furthermore, a large enough set of *workers* may also temporarily saturate the rendering task assignment mechanism of the *manager* to control which nodes will be assigned [12]. These nodes already know the verification keys used for the assigned tasks, then they generate straightforwardly forged images which validate the noise verification. In short, once the verification key generation is predictable, the noise verification will be compromised.

This attack is mitigated due the cryptographic hash property of $K_{\texttt{verif}}$ (c.f. proposition 3): knowledge about generated keys does not leak any information about the next keys.

2) Attacks on noises: Another kind of attack is based on analyzing rendered frames to predict the possible positions of distortion regions. Below is a simple result for extreme cases.

Proposition 4. *Let $I = \mathcal{R}(S)$ be the rendered frame of some scene $\mathcal{S}$, if I is a constant signal or white noise then ANGV scheme is perfectly secure.*

Proof. We prove for the case of constant signals, the argument for white noises is similar. For simplification, we do not take the constraints about the fidelity of watermarked signals in to account and suppose that I is the constant binary signal $I(x, y) = 0$. The distortions of any noise vector of length n occurs at distinguished and uniformly random positions $(x_i, y_i)_{1 \le i \le n}$ on the image, or $\hat{I}(x_i, y_i) = 1$ for all $1 \le i \le n$. Since $K_{\texttt{verif}}$ is a cryptographic hash, $P_C = P_S = \mathcal{U}_I^{\otimes n}$, hence $D(P_C \parallel P_S) = 0$. □

Many researchers observe that the data watermarking can be considered as the communication over noisy channel where the watermarks are signals and the content data is noise [8], [21]. Under this perspective, proposition 4 is actually a special case of the Shannon's noisy channel coding theorem. The spectrums of signals are Dirac pulses for constant signals and white noises for white noises, then noises can be indistinguishably inserted everywhere. From the attacker's point of view, there is no information to make any significant estimation about the positions of the watermark.

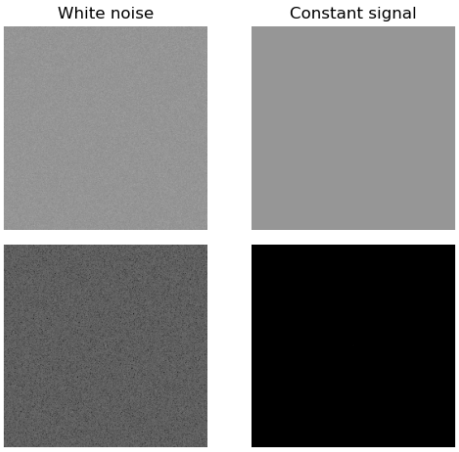


Figure 8: Trivial signals and magnitude spectrums

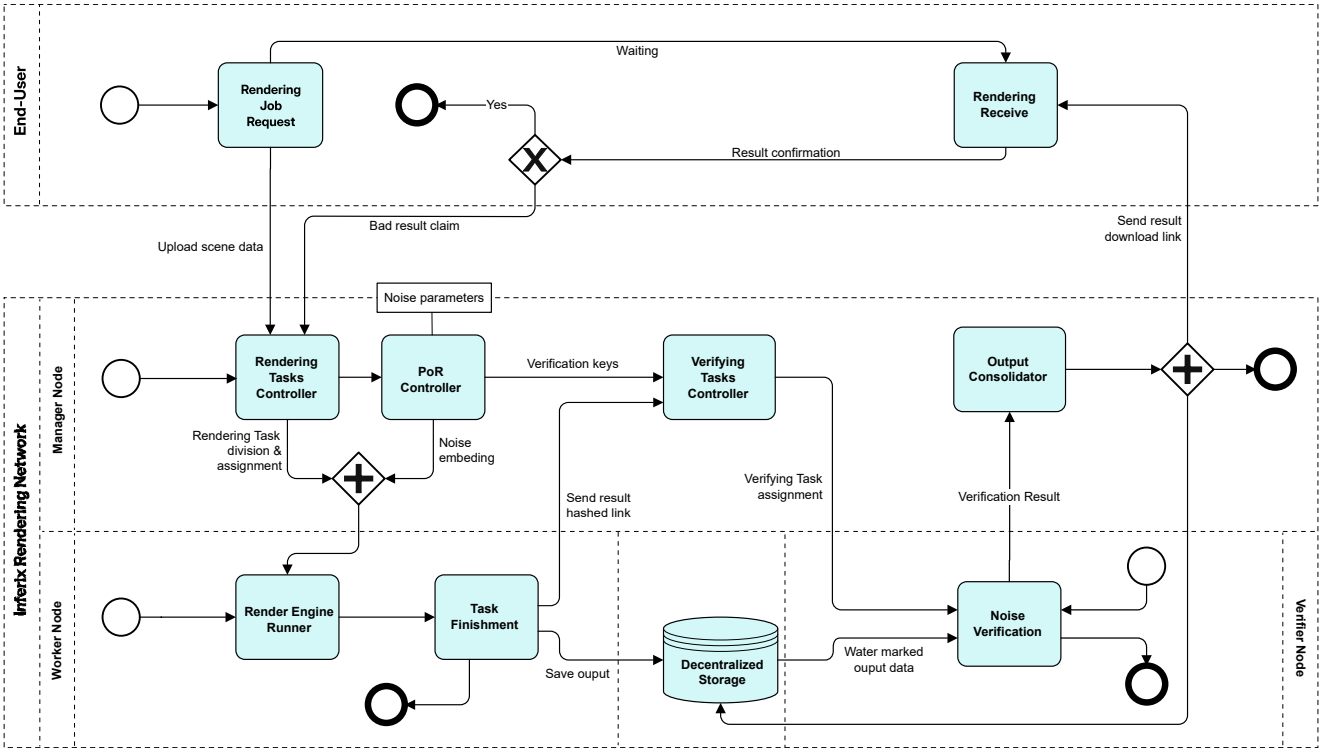


Figure 7: Rendering flow with Active Noise Generation and Verification

Scene	Rendering time	Rendered frame size
Coca-Cola	8.09	1080×1080
Grease Pencil Bike	4.30	2880×1620
Blender 3.5 Splash	11.41	1327×1250
Bathroom Above Corner	146.41	4000×3000

Table I: GPU rendering time of scenes (in seconds)

3) Attacks on verifiers: In case where *verifiers* are compromised, we employ the consensus mechanism of the Inferix blockchain network: the verification will be executed by several *verifiers* nodes through consensus. In order to minimize computational resource wastage, once two or more nodes reach a consensus, then the rendering result is considered successfully verified.

G. Performance evaluation

We evaluate the execution time of noise insertion and noise verification on several scenes, the lengths of the noise vectors variate from 2 to 40. The tests are executed on a workstation of Intel®Core™i5 2.5 GHz CPU, 32 GB RAM and NVIDIA GeForce RTX 3070 GPU. The results are given in fig. 9 for noise insertion and in fig. 10 for noise verification, detailed data is given in tables in appendix D.

The noise insertion needs to analyze the structure of the input scene to generate and insert noises, that explains the experimental results in fig. 9 where the execution time, while being proportional with the length of the noise vector, depends importantly on the complexity of input scenes. A loose quantification for this complexity can be observed via the execution times needed to render the scenes, shown in table I.

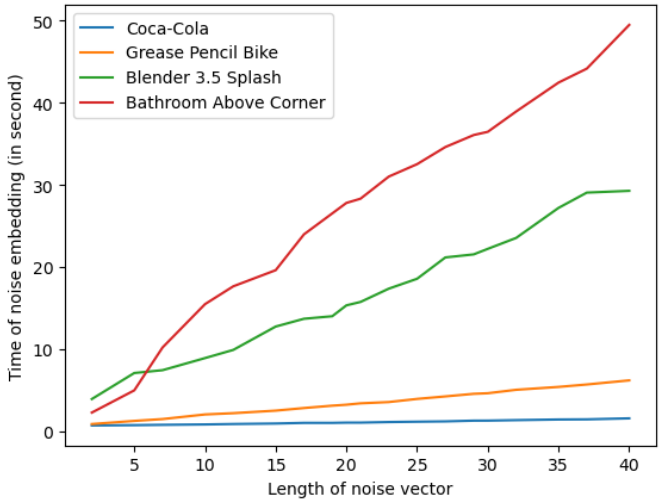


Figure 9: Noise insertion

Whereas the noise verification needs only to analyze the distortion regions whose locations are given by the verification key, then the execution time depends mostly on the number of the regions (which is also the length of the noise vector) and slightly on the size of the rendered frame.

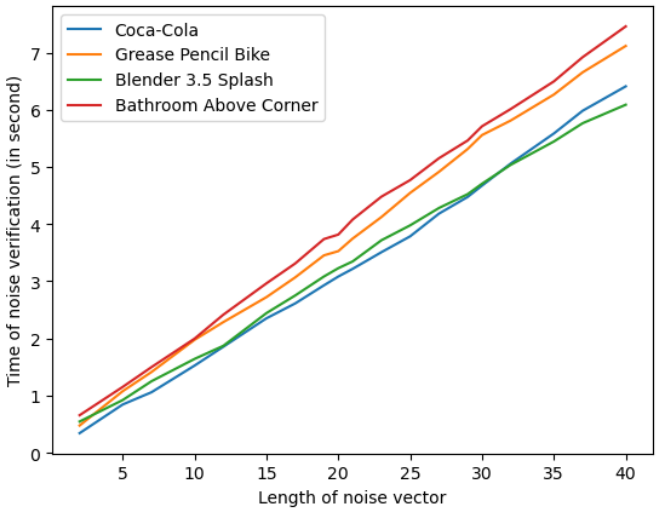


Figure 10: Noise verification

H. Integration

We integrate the Active Noise Generation and Verification scheme into the original rendering flow (c.f. fig. 1) by placing respectively the noise generation and the noise verification into the rendering task controller of the *manager* and the proof-of-rendering verification of the *verifier*. The completed flow is depicted in fig. 7.

In the introductory section of the paper, we have discussed the problem of rendering verification, that is to automatically verify whether the submitted scenes of users are genuinely rendered or not. The ANGV is proposed to deal with the challenge, ANGV serves then as a *proof of rendering* (PoR), inspired from the *proof of ownership* schemes [2], [3]. Other components of the Inferix network simply refers PoR for the underlying ANGV algorithm.

IV. Decentralized visual computing

Inferix is a decentralized physical GPU network connected to end users, graphic software, or AI models through a feature-rich software layer that is continuously expanded by Inferix Labs and the community of developers within the Inferix ecosystem. Inferix is built upon the core PoR algorithm, which integrates both on-chain and off-chain verification. This section will outline the system architecture and key design details of the Inferix decentralized GPU system.

The system architecture of the Inferix network is described in fig. 11, consisting of three main components: Manager Node, Worker Node, and Client Apps. Data is stored and accessed through a Decentralized Storage System.

A. Client Apps plugin

Inferix provides plugins and APIs that allow traditional graphic design software to easily send 2D/3D graphic data into the Inferix decentralized rendering system and receive photorealistic images or videos in return.

At the time of writing this paper, there is only one project on the market offering a GPU-based decentralized rendering solution, but it requires users to use (and pay for) their proprietary

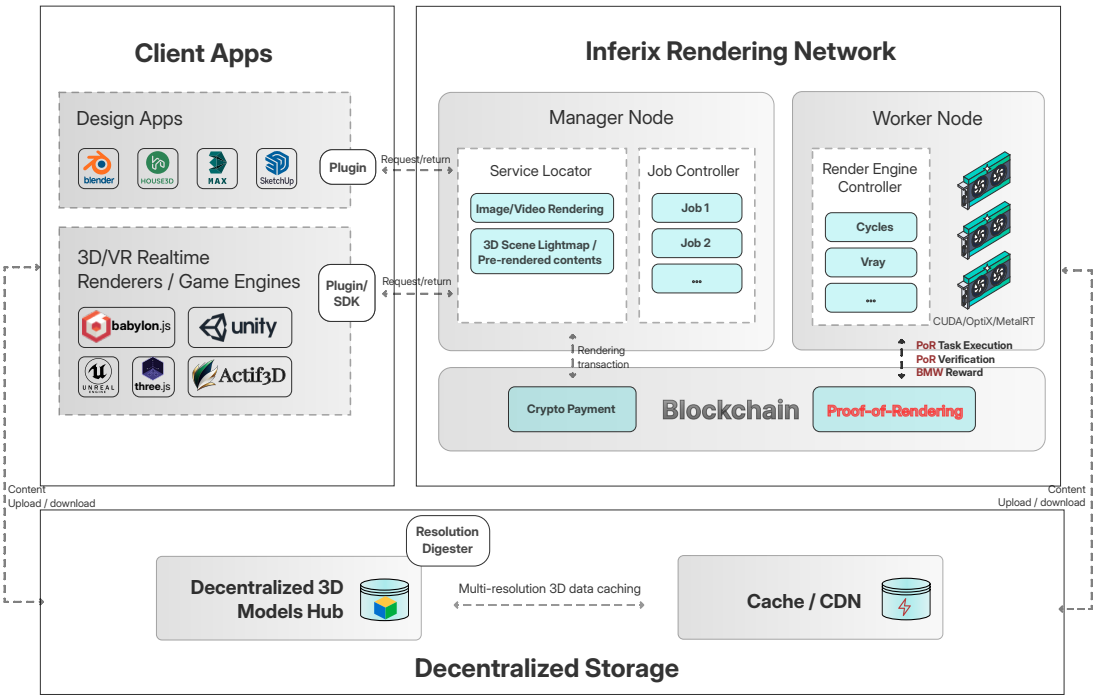


Figure 11: Inferix decentralized rendering architecture

rendering engine. Inferix emphasizes the principle of freedom in line with the Web3 spirit, not mandating that end-users use a specific rendering engine or graphics software to leverage its GPU compute network. The Client App Plugin system, developed collaboratively by the Inferix team and developers within the ecosystem, supports most major rendering software and engines such as Blender, SketchUp, Cinema 4D, 3ds Max, and more.

B. Client API and SDK

The core of all real-time rendering engines like Unity, Unreal Engine (UE), Three.js, and Babylon.js, Actif3D [22] lies in the combination of static space rendering (lightmap baking) and real-time rendering of dynamic elements. Lightmap baking is typically performed by developers during the game build process, but this process often requires several hours and expensive hardware. Traditionally, a large amount of CPU power was used for this task, but recently, most game studios have shifted to using GPUs. However, the costs associated with GPUs remain high, and the long rendering times result in significant waste and expense.

Inferix offers a decentralized infrastructure for baking lightmaps at a lower cost. Moreover, by leveraging parallel processing across multiple hardware setups in different locations, Inferix can significantly reduce rendering times for large-scale projects. To support this process, Inferix provides tools for lightmap baking through its rendering system, along with an SDK that enables the integration of these baked lightmaps into various rendering engines.

C. Manager Node

Manager Node(s) are computers that handle API load balancing and manage Inferix's services, including Rendering, AI Training/Inference, and Remote PC services.

A manager node consists of two components: Service Controller and Job Controller. The rendering cost calculations based on the PoR algorithm will be sent by the Service Controller to the end users, allowing them to decide whether to submit an order. The Job Controller (c.f. section I-A) is responsible for dividing rendering jobs into different tasks and assigning them to Worker Nodes for execution. For example, a 1000-frame video rendering job can be divided into 200 tasks, with each task rendering 5 frames. These tasks are pushed into TaskQueue. Available workers that meet the minimum hardware capability requirements, based on the Inferix Bench index (c.f. section VI-E), will be randomly assigned to perform the rendering tasks. The rendered results are then aggregated according to the process outlined in fig. 1.

D. Worker Node

It is where the actual rendering takes place. Each Worker Node is equipped with a tool called the Render Engine Controller, along with one or several render software or render engines, such as Blender Cycles, V-Ray, Keyshot, D5 Render, Octane, etc. See fig. 1 for more details.

The render engines utilized by Worker Nodes typically rely on path-tracing/ray-tracing techniques, which demand substantial compute resources. Some of these engines are open-source and free, such as Cycles, while others, like V-Ray, are paid software. If there are any render engine costs, they will be factored into the rendering price that the Inferix network charges users.

E. Decentralized Storage and Data Security

Data storage and security are critical for Inferix users. The data for a 3D scene typically ranges from a few dozen MBs to several GBs, while AI model data can reach up to several TBs. Managing this data requires specialized methods.

1) Data categories: 3D data stored in the Inferix system includes:

- Geometry data of 3D models, characterized by polygons that form the shape of objects. Depending on the render engine, different formats may be used. The higher the number of polygons, the more detailed the rendering result will be; however, the render time will also increase, and more storage space will be required.
- Texture data, which is the surface image of the object. Inferix uses data formats with the best compression algorithms for GPUs, such as Basis and KTX, alongside common formats like JPEG, PNG, TIFF, or WebP.
- Rendered results in image format
- Rendered results in video format
- Structural data in JSON format

2) Multi-level 3D polygon data: In the stored data on Inferix, aside from images and videos, the geometry data of 3D models consumes the most storage space. Each time a render is performed, the Worker Node must download this data locally so that the render engines can execute the task. This process can consume significant bandwidth and time. To save bandwidth and reduce download times, 3D models are converted into two levels of detail, known as high-poly and low-poly, and are pre-stored on the Inferix network. The existence of multiple levels of 3D data is referred to as *multi-level 3D polygon data*.

3) Polygon digester: After a 3D data file is stored on Inferix, it may be queried multiple times by Worker Nodes or design software. Depending on the needs of the query, the original data is converted into different levels of polygon detail through a lossy conversion algorithm. This process is handled by the *Polygon Digester* tool within the Inferix storage system. This tool ensures that the appropriate level of detail is provided for each task, optimizing both storage and performance by reducing unnecessary complexity in the 3D models when high detail is not required.

4) Decentralized storage: With the large volume of data involved, using traditional cloud storage models at traditional data centers can be extremely costly. Peer-to-peer (P2P) and decentralized storage networks like IPFS and Filecoin offer a significant reduction in costs while maintaining access speeds comparable to traditional methods. The 3D data storage system of Inferix will predominantly rely on such decentralized networks, leveraging their cost-effectiveness and efficiency for managing extensive data volumes.

5) Decentralized cache: Inferix's 3D data caching system utilizes decentralized CDNs. This approach enhances the distribution and retrieval of data across the network, reducing latency and improving access speeds by caching frequently requested 3D assets closer to the end-users. The decentralized nature of the CDN ensures that caching is distributed across multiple nodes, providing redundancy and resilience while minimizing the load on any single server. This setup aligns with Inferix's broader strategy of leveraging decentralized technologies for efficient data management.

6) Data security: Data stored within the Inferix network is categorized into two types:

- *Session data*, which includes input data along with temporary data generated during the rendering process. This data typically exists for a short duration, ranging from a few minutes to several hours.
- *Persistent data*, which consists of the output from the rendering process and is stored long-term in the system. Examples of persistent data include images and videos after rendering, or VR scenes created after lightmap baking.

Inferix encrypts session data to ensure it remains secure against decryption attacks during data transfer. For persistent data, Inferix offers long-term hosting on its storage system and allows users to share the data publicly or with specific permissions over the internet.

V. Decentralized federated AI

In section IV, we introduced the physical GPU network used for graphics rendering. Next, we will discuss how we utilize this GPU network for AI training and inference. These processes are essential items of Inferix's Phase 2 strategy, aligning with the core principles of Web3: openness, decentralization, self-governance, and diversity.

Currently, AI advancements are predominantly driven by industry giants like Google and OpenAI, relegating most users to passive roles. This situation runs counter to the principles of Web3 and DePIN. To bridge this gap, we propose an application framework for deploying federated learning models on the Inferix GPU infrastructure in the following sections. This framework is designed not only to reshape the existing landscape but also to elevate the intelligence of the evolving DePIN ecosystem.

A. Federated learning with TensorOpera

Federated learning and its practical benefits have recently started to see widespread application. This article will not delve into the concept of federated learning itself but will focus on applying it to leverage the GPU infrastructure of Inferix.

Several foundational projects have developed tools/SDK for federated learning developers. After extensive evaluation, we have chosen the open-source TensorOpera® as the basis for developing the Inferix Federated Learning framework.

B. Meta LLaMA

In its GPU hardware segment, Inferix focuses on devices optimized for graphics rendering, with the RTX3090 and RTX4090 serving as the flagship devices.

The TensorOpera® team has released public data on deploying pre-trained models like LLaMA-2 13B or LLaMA-3 7B parameters on the RTX4090. Notably, LLaMA-2 13B inference running on a single RTX4090 using TensorOpera's ScaleLLM achieves 1.88 times lower latency compared to the

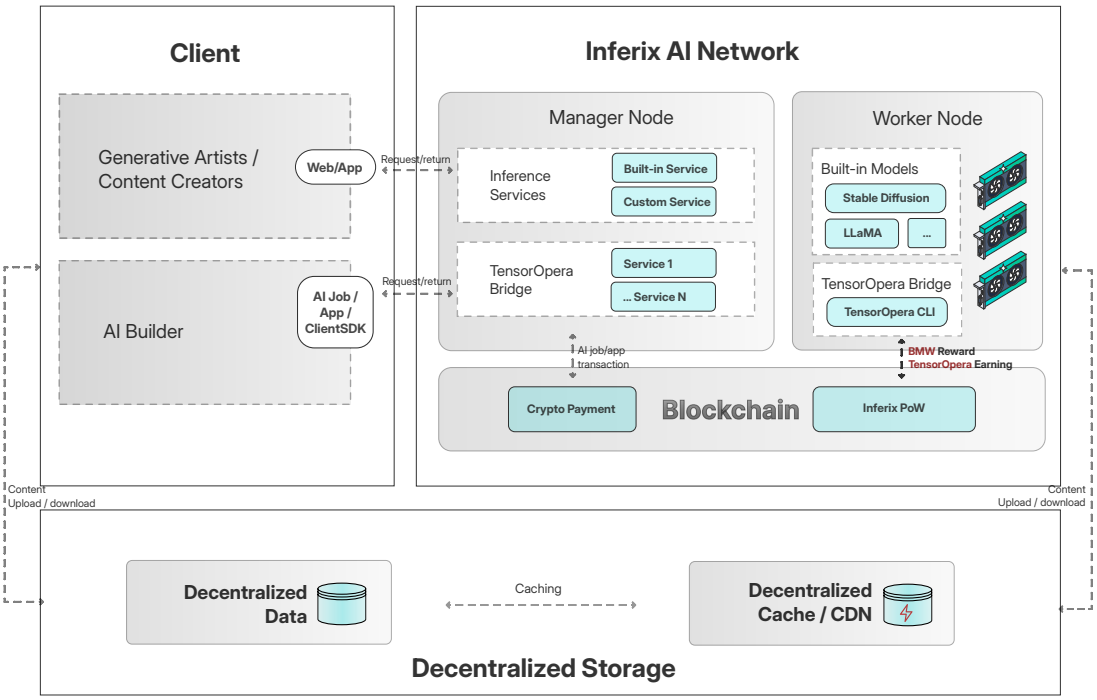


Figure 12: Inferix and TensorOpera integrated architecture

same model running on a single A100 GPU using vLLM. For the LLaMA-3 7B, it can run with a token batch size of 256 on a single RTX4090, without additional memory optimization [23].

In their introduction to ScaleLLM, the TensorOpera® team claims that by utilizing this engine with the RTX4090, LLMs can operate with three times less memory, run 1.8 times faster, and be 20 times more cost-effective compared to using A100 GPUs in traditional data centers.

Research and experimental benchmarks have shown that we can *train larger LLMs on a larger number of distributed GPUs than in data centers with federated learning*, using Gradient Low-rank Projection (GaLore) [23], [24].

C. Stable diffusion

Stable Diffusion inference can be easily deployed on various Inferix hardware models, ranging from the RTX3070 to the RTX4090 nodes. Additionally, the RTX4090 node is an exceptionally well-suited hardware model for training Stable Diffusion models.

D. Other AI models

Other popular AI models, such as Bark, InstantID, and Whisper, both training and inference, can also run on Inferix GPUs through the TensorOpera AI platform.

E. Inferix AI

From the information presented above, we can conclude that the hardware of the Inferix network is well-suited to serve as an infrastructure for federated AI. Next, we will discuss the design of the Inferix Federated AI system.

In the architectural design (c.f. fig. 12), Inferix enables *generative AI artists* and content creators to access AI models trained by *AI Builders* within the Inferix community, as well as models trained by the Inferix Team itself (built-in models). These services can be hosted on the Inferix Manager Node system or on the TensorOpera AI platform.

AI Builders have the option to run their models directly on the Inferix infrastructure or through the TensorOpera Bridge. The output result can be hosted in Inferix infra with custom domain option.

In addition to handling graphics rendering tasks, Inferix GPU Nodes also serve as Federated Learning Clients by running the Inferix TensorOpera CLI.

- *TensorOpera CLI:* the CLI client that is built based on TensorOpera open source with Inferix PoW algorithm integrated.
- *Inferix PoW:* general PoW algorithm used to calculate the actual work performed by workers, excluding those involved in rendering tasks. Inferix PoW is based on the Proof-of-Rendering mechanism to calculate the Inferix Bench (c.f. section VI-E), incorporating an algorithm to accurately measure the actual working time of a node.

VI. Economic Model

A. GPU compute market for visual computing and federated AI

The Visual Computing market was valued at over USD 36.5 billion in 2023 and is projected to grow at a CAGR of more than 23% from 2024 to 2032. A significant trend within this market is the rise of cloud-based rendering services, which offer substantial benefits to designers, content producers, and businesses alike [25]. The global federated learning market

was valued at USD 110.82 million in 2021 and is expected to grow at a CAGR 10.7% during the forecast period 2022-2030 [26].

There are approximately 4 million graphic designers working globally, generating an average of about 1.4 billion rendering tasks per year. Traditional 3D graphic design applications such as Blender, SketchUp, 3ds Max, Maya, Cinema 4D, House3D, Actif3D... all require rendering operations to produce images or videos that accurately simulate spaces, characters, animations, and materials as they are designed. Currently, designers using these software tools often need high-end PCs with expensive GPUs. For example, an interior designer often uses a computer worth around $2000. Each 3D video rendering process usually takes several hours. In terms of software, aside from investing in 3D modeling software, users also need to use a specialized software called a rendering engine, such as VRay (paid) or Blender Cycles (free).

There is currently only one decentralized rendering solution on the market, which serves as a competitor to Inferix. However, this solution requires users to utilize a proprietary rendering engine developed by them, which comes with a significant licensing fee.

With Inferix, users simply need to install a plugin into their preferred 3D software to access a crowdsourced GPU network, allowing them to submit rendering requests with ease. This approach significantly reduces rendering time compared to using a standard PC and offers substantial cost savings compared to traditional render farm services on the market.

B. Inferix vision

Inferix plans to implement a token economy model to incentivize individuals and organizations worldwide to share their GPU compute resources. This approach is designed to not only lower the cost of GPU resources and enhance efficiency but also create new revenue opportunities and business models for participants in the decentralized network.

Inferix envisions creating a fairer, more efficient, and sustainable distributed computing ecosystem through innovative technology and economic models. By challenging the traditional monopoly on GPU power, Inferix aims to promote the equitable distribution and efficient use of GPU resources, fostering the broader adoption and development of visual computing, AI, and blockchain technologies.

C. $IFX Token

Inferix tokens ($IFX) are integral to the value exchange within the Inferix ecosystem and the operation of its decentralized GPU network. GPU owners can earn $IFX tokens through the following methods:

1) Participation in Inferix visual computing: Participating in the Visual Computing network is one of the primary ways to earn $IFX token rewards. Participants contribute their computational resources (CPU/GPU, internet bandwidth, storage) to assist with graphic rendering tasks. This includes not only performing rendering work but also contributing to PoR verification. The greater the contribution, the greater the $IFX token reward.
2) Contributing to Inferix federated AI tasks: Participating in the Federated AI network is another key way to earn $IFX tokens. In this process, participants contribute their computational resources to support the training or inference of AI models. Additionally, individuals or groups can earn $IFX token rewards by engaging in activities such as optimizing models and improving training efficiency.
3) Running Verifier Node: The Verifier nodes ensure the integrity and service quality of Inferix network by checking the visual computing or AI workers and their service process. The parameters mainly include liveness, capacity, and quality of service. The checking methods include heartbeat collection, benchmark testing using PoR, link data collection and analysis.
4) Engaging in Inferix Governance activities: Another way to earn $IFX tokens is by participating in the governance of the Inferix network. $IFX token holders can take part in the decision-making process, such as voting on network protocol updates and adjustments to BMW reward parameters. This participation not only enhances the network's democracy and transparency but also allows holders to directly influence the direction of Inferix. Governance participants are rewarded with $IFX tokens based on their level of contribution to the network's decision-making. This approach is designed to incentivize and empower those who actively participate in Inferix governance, ensuring that the network evolves for the common good of the community.
5) Joining Inferix Professional Artist Network: 3D artists, designers, and architects can join the Inferix Professional Artist Network and utilize Inferix GPU infrastructure services. Based on the volume and frequency of each individual's IBM (c.f. section VI-E) consumption, Inferix will implement an $IFX token reward mechanism to encourage end-users to use the service.

For GPU owners, please refer to section VI-I for more details on how to earn rewards by running an Inferix node.

D. Burn-Mint-Work token issuance model

Increasing the *token velocity* and controlling inflation are critical issues for any utility token. Various issuance models for utility tokens have been proposed, but fundamentally in DePIN projects, there are two commonly used methods: *Work Token* and *Burn-Mint-Equilibrium* (BME). While the Work Token (used by Filecoin project) has the advantage of improving *token velocity*, it requires the provided service to be purely a commodity, with no manual human intervention. On the other hand, BME (used by famous projects like Helium and Factom [27], [28]) lacks control over the volume of verified work completed and the ability to adequately penalize substandard service providers [29].

Inferix's BMW (Burn-Mint-Work) is the token issuance mechanism designed to address the creation of tokens based on the amount of work completed and a Node's working capacity. When BMW is combined with the IBME mechanism (c.f. section VI-E), it also helps solve the inflation control issue

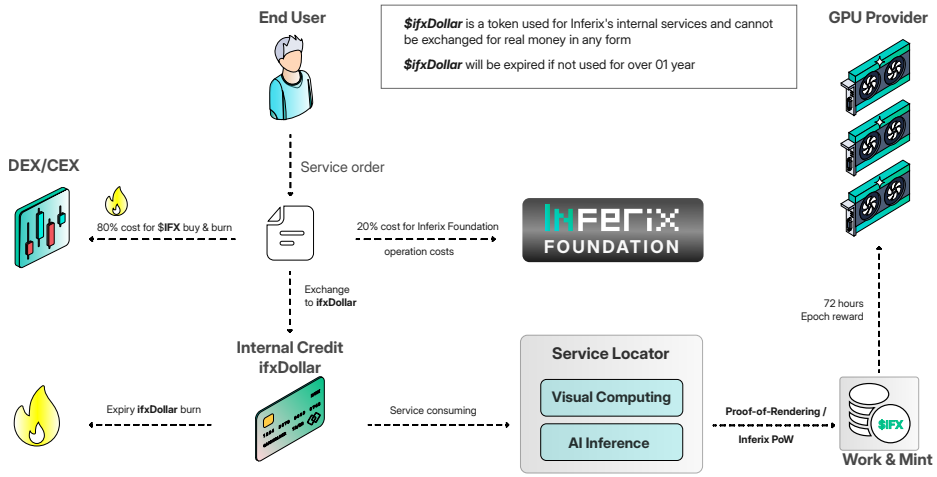


Figure 13: Inferix Burn-Mint-Work token issuance model

within the Inferix system, allowing $IFX tokens to be minted flexibly based on the total volume of work completed and the total amount of money users pay for services across the Inferix network.

Inferix's BMW is an improvement on the BME algorithm, with the important parameter being "work" calculated based on the PoR algorithm. While BME (Burn and Mint Equilibrium) balances only two parameters, burn and mint, BMW balances three parameters: burn, mint, and work. BMW also incorporates penalty mechanisms for substandard providers from the Work Token model.

When Inferix completes a graphics rendering or federated AI task t for a customer, by the Provider A, the network charges a service fee calculated as follows:

$$\mathcal{F}(t) \triangleq W_t \times \mathcal{P}_A \tag{17}$$

where W_t is the amount of work completed for t measured in IBM (c.f. section VI-H), and $\mathcal{P}(A)$ is the unit price for one IBM, which can be adjusted by Provider A. The Matcher algorithm on the Manager Node automatically searches and provides the best options for both the user and the provider.

To order services, users must top up a prepaid account with Inferix's internal payment token, called *ifxDollar*, at an exchange rate of 1 USD = 1 ifxDollar at the time of the top-up.

After completing a rendering job, 80% of the service fee is used to purchase an amount of $IFX tokens from available supply sources (e.g., DEX/CEX exchanges). This amount of $IFX tokens is then burned. Next, 20% of the service fee will be retained and managed by the Inferix Foundation, these funds will be used to reward developers who contribute to the Inferix ecosystem.

The issuance of new tokens on Inferix is executed after each *epoch period*, initially set at 72 hours but later adjustable through DAO governance. Suppose after an epoch period p, the total amount of work completed across the network is W_p. In that case, the system will issue a quantity of tokens according to the following formula:

$$\mathcal{T} \triangleq W_p \times E_p \tag{18}$$

where E_p is a parameter taken from the Emission Plan, which is planned by Inferix Governance and periodically adjusted by monitoring the IBME ratio (c.f. section VI-H).

The newly issued $IFX tokens are allocated to stakeholders in the network as detailed in section VI-G. This token issuance process must be entirely independent of the token burn process to generate *ifxDollar* mentioned above. When service demand increases, more $IFX tokens are burned, leading to a decrease in the total supply of $IFX, which puts upward pressure on the price of $IFX tokens. This price increase results in fewer tokens needing to be burned to complete the same amount of work, thereby bringing the system back into equilibrium. An increase in the price of $IFX tokens also increases the profitability of providers, attracting more providers joining and increasing supply. When service demand decreases, the opposite scenario occurs, leading to a state of equilibrium.

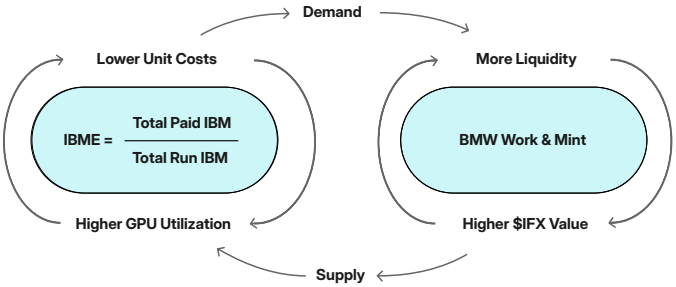


Figure 14: IBME

E. Inferix bench and IBME

1) IB and IBM: Based on PoR, Inferix can assess the computing power of a Worker Node at a given time and uses a measurement unit called *Inferix Bench* (IB). When IB is multiplied by the node's working time, it results in *Inferix Bench minutes*, abbreviated as IBM. Thus, IBM is the metric used to measure the workload within the Inferix network.

To provide a quantitative perspective, 1 IB is defined as the average rendering capacity of a *standard unit node* with 2x

RTX 4090 GPUs, 32 GB 32GB of RAM, 1x Intel®Core™ i9 CPU, SSD storage. This figure is updated daily using DAO mechanism, using benchmarks of a set of sample scenes on 10 nodes. The hardware specification of *standard unit nodes* and the *sample scenes* are also DAO-adjustable.

Assuming the average rendering time for one frame of a scene G in the sample set is T_G^0, then it is not a fixed number, but is instead derived from the combined rendering capacity of the 10 randomly selected *standard unit nodes* at the benchmarking time. The value of T_G^0 is influenced by GPU, CPU, storage read/write speeds, and network speeds at the time of benchmarking, though the variation is negligible.

To determine the IB of any given node n, Inferix sends render requests for scene G (randomly selected) to that node periodically. Assuming the average time it takes that node to render one frame in G is T_G, the rendering power of n is defined by:

$$\mathrm{IB}\,(n) \triangleq \frac{T_G^0}{T_G} \tag{19}$$

Thus, the larger the T_G, the smaller the IB_n value.

2) IBME: short for *Inferix Bench Minutes Efficiency*, is an index used to evaluate the working efficiency of a set of nodes $\mathcal{N}$ over a working period P, determined by the formula of the total IBM paid over the total apparent IBM:

$$\mathrm{IBME}_P\,(\mathcal{N}) \triangleq \frac{\sum_{n\in\mathcal{N}} \mathrm{IBM}_P\,(n)}{\sum_{n\in\mathcal{N}} \mathrm{IBM}_P^a\,(n)} \tag{20}$$

where $\mathrm{IBM}_P\,(n)$ is the customer paid IBM for node n and $\mathrm{IBM}_P^a\,(n)$ is the apparent IBM of node n calculated by multiplying the IB of a node by the elapsed time (in minutes) of period P. We observe that IBME is always a figure below 100% and the higher the IBME ratio, the more efficiently the network operates.

Recent reports from DePIN projects have highlighted a significant issue: while the number of nodes participating in the network is very high, many of these nodes are either inactive or active but not receiving any service requests. By controlling the IBME, Inferix will solve this problem.

In addition to planning and adjusting the Emission Plan (c.f. section VI-D), IBME also assists Inferix Governance in making decisions related to the use of the Inferix Foundation's funds for promotional activities aimed at increasing supply or attracting service demands.

F. Price simulation

Now, we will build a price simulation for Inferix's GPU rendering service and compare Inferix's price competitiveness with traditional Cloud Rendering services. The details of these calculations are listed in appendix B. The table in fig. 15 shows the most important information.

In this simulation, the calculations are standardized for a single GPU device. The *Provider Price* refers to the minimum service price that a GPU Provider will set to ensure that revenue is always twice the cost per hour of operation.

In the best-case scenario, where GPUs receive enough tasks to operate full-time every day, providers can break even in no more than 17 months for the RTX 3060, RTX 3070, and RTX 3080 models, and 24 months for the RTX 3090 and RTX 4090 models. However, since the RTX 3090 and RTX 4090 models are better suited for federated AI tasks, their IBME scores may be higher. Moreover, the choice of which GPU model to use for rendering also depends on whether the customer prioritizes shortening the rendering time or prefers a more reasonable price. Therefore, the decision on which device to invest in depends on the specific conditions of each individual in particular market conditions.

GPU Model	Provider Cost ($/h)	Provider Price ($/h)	Provider Profit ($/h)	Provider Payback Time (months)	Inferix Cost ($/h)	Inferix Price ($/h)	Traditional Competitors Price
RTX3060	0.050	0.100	0.050	16	0.155	0.186	N/A
RTX3070	0.067	0.134	0.067	17	0.189	0.227	N/A
RTX3080	0.107	0.214	0.107	19	0.269	0.323	0.830
RTX3090	0.138	0.277	0.138	24	0.332	0.398	1.940
RTX4090	0.176	0.352	0.176	24	0.407	0.488	3.800

Figure 15: Pricing simulation

A crucial point to note here is that with the proposed *Provider Price*, Inferix's service price is the equivalent of only from 12% to 38% of the average price in the traditional cloud rendering market at the time of writing this paper [30], [31], [32]. This means that providers can adjust the price higher to shorten the payback period.

Price of 1 IBM: Based on the calculation method for IBM discussed in section VI-E, the price of 1 IBM is approximately $ 0.16 according to this pricing simulation.

G. Token metrics and allocation

Token: INFERIX
Ticker: $IFX
Max Supply: 1,000,000,000 $IFX

The total supply of $IFX is determined based on the assumption that the Inferix network will serve approximately 4 million graphic artists and around 10 million end-users utilizing generative AI services. With 1 billion $IFX tokens in circulation, the *token velocity* of Inferix is expected to reach a minimum of 10.0, according to the Equation of Exchange $M \times V = P \times T$.

1) Token allocation: The token allocation of $IFX is depicted in fig. 16, that includes:

- ***Investors:*** Inferix raises capital from investors, including Seed, Strategic, KOLs, and Public Investors, using 22.5% of $IFX total supply.
- ***Liquidity and listing:*** 0.6% for DEX and CEX.
- ***Community grants Fund:*** 14.4% for:
  - initial airdrop for OG members,
  - initial grants for GPU providers,
  - education grants for using Inferix in schools.
- ***Ecosystem fund:*** 50% for:
  - staking reward,
  - PoR reward using BMW,
  - Inferix platform expansion projects,

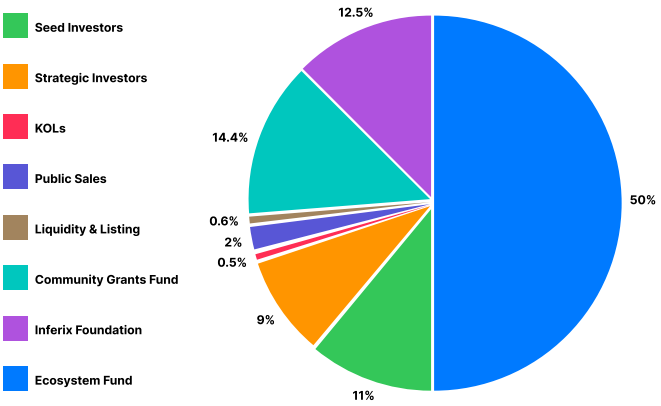


Figure 16: $IFX token allocation

  - Inferix hackathon and bug fixes,
  - Inferix DAO building.
- ***Inferix foundation:*** 12.5$ for:
  - Inferix Labs operation activities,
  - Inferix global expansion.

2) Token vesting: The token vesting is show in table II.

Seed	6 months cliff, then 12 months linear vesting
Strategic	6 months cliff, then 12 months linear vesting
KOLs	10% at TGE, 3 month cliff, 15 months linear vesting
Public Sale	10% at TGE, 3 months cliff, 4 months linear vesting
Liquidity and listing	100% unlocked at TGE, circulated when listed
Community Grants Gund	Unlock 3 million IFX at TGE for community airdrop. Later schedule is under DAO governance
Ecosystem Fund	Locked at TGE, gradually unlock under the BMW model for GPU providers and Ecosystem partners
Inferix Foundation	12 months lock, linear vesting 24 months

Table II: Token vesting

H. Governance

$IFX token holders participate in the network's governance decisions, including protocol updates, reward policy adjustments, and so on. This decentralized governance structure ensures the democratic and transparent nature of the network. Governance decisions are made through a weighted voting mechanism where $IFX holdings determine the voting weights. This encourages long-term holdings and active participation in network governance.

The Inferix Network backend infrastructure is built on three core components: Worker Node, Manager Node and Verifier Node. With the expectation of serving 4 million designers, Inferix will have around 1 million Worker nodes, with a significant portion coming from the PCs of the designers themselves. In a scenario where 4 million render jobs are processed daily, the Inferix network would require approximately 2500 Manager Nodes to efficiently coordinate and handle the PoR algorithm. In this scenario, Inferix would need approximately 25,000 Standard Verifier Nodes to ensure the entire network operates seamlessly with low latency. A standard verifier node is a PC that meets standard verifier hardware requirements (c.f. appendix C).

I. Node staking and rewards

Inferix allocates up to 50% of its total token supply as rewards for its node system, sourced from the Ecosystem Fund (c.f. section VI-G2). Additionally, Inferix will distribute 80% of its service revenue to the nodes through BMW model (c.f. section VI-D). To earn these rewards, node owners must stake a certain amount of $IFX tokens through purchasing a node license and actively run their nodes to participate in the Inferix network. Below is the reward mechanism for the nodes in the Inferix network:

1) Worker: Workers are the most critical component of the Inferix network. The reward pool for Worker Nodes constitutes 75% of the Ecosystem Fund and is distributed through the BMW model. In other words, three-quarters of the 80% of the network's service revenue is allocated to Workers.

To receive BMW rewards, each Worker must hold an amount of $IFX tokens at any given time. This token amount acts as a penalty fund in case the Worker fails to meet Inferix's standards. The minimum stake amount of node N can be calculated as follows:

$$S_N \triangleq 2 \times \mathcal{E} \times \mathrm{IB}_N \times P_N \tag{21}$$

where IB_N is the benchmark value of the node (c.f. eq. (19)), P_N is the price per IBM defined by the node provider himself and $\mathcal{E}$ is a constant defined by number of minutes of the Epoch period, that is $72 \times 60 = 4320$.

For example, if a provider owns a node with 32 GB RAM, Intel®Core™i9 CPU, and an RTX4090 card, its computing power will be approximately 0.5 IB. To participate in the Inferix network at a service price of $ 2 per hour, it is needed to hold a quantity of $IFX tokens equivalent to $0.5 \times 2 \times 4320 \div 60 = 72$ USD.

Inferix Edge: computers are specially designed to participate in the Inferix worker network. These machines are streamlined by removing non-essential components and focusing on enhancing the CPU, GPU, RAM, and motherboard to optimize performance for Visual Computing and federated AI tasks.

2) Verifier: Verifier is also an important component in the Inferix network and is allocated 7.5% of the total $IFX token supply for the reward pool. Inferix mobilizes a total of approximately 25,000 Verifier nodes through node sale programs. The first node sale program will be announced before TGE. In order to run a Verifier node, we do not need to invest in powerful hardware like Worker nodes, it is possible to run the verification on a mobile device (c.f. appendix C).

3) Manager: The Inferix network requires 2500 Manager nodes, which will be sourced through Manager Node License sales. Initially, when the mainnet goes live, the *Inferix Foundation* will deploy no more than 100 in-house nodes of this type to ensure network availability. The plan for Manager node sales will be announced before the TGE. The reward pool for Manager Nodes is 1.25% of the total supply of $IFX tokens.

4) Node Sale and Penalty Pool: Node Sales are crucial programs for building Inferix's compute network. Participants in these programs are required to stake a certain amount of $IFX tokens by purchasing a node license. The number of $IFX tokens needed varies depending on the type of node and the timing of the license purchase. Node Sale programs will be organized into tiers based on the order of the sale. As a general rule, the earlier the license is purchased (the lower the tier), the cheaper the price. Licenses purchased before the TGE are referred to as Whitelisted licenses. Participants who acquire Whitelisted licenses will receive more benefits and discounts compared to those who purchase after the TGE.

Penalty Pool is a token fund collected from penalties imposed on Worker Nodes that violate Inferix standards while performing assigned visual computing or AI inference tasks. The Penalty Pool is used as a reward for Verifiers who actively contribute to ensuring the efficient operation of the network.

VII. Future development

A. PoR and NFT minting for graphics creative assets

During the process of creating their work, graphic artists frequently engage in rendering activities. The results and traces generated during rendering on Inferix can be used as proof of artistic creation. Naturally, PoR combined with native blockchains can be utilized to mint NFTs that fully capture the artist's creative process. In the future, the Inferix ecosystem could potentially host projects related to this field.

B. ZKP and PoR combination

Zero-knowledge proofs (ZKP) have recently garnered significant attention from developers in the web3 space. At present, applying ZKP to heavy edge computing systems, such as GPU-based rendering or AI, faces challenges due to the high computational resources required, leading to reduced processing performance. However, the Inferix development team believes that ZKP will become increasingly valuable for our use cases in the future. Therefore, we are continuing to explore experimental approaches that combine ZKP with PoR. We will publish these experiments in future papers.

C. Inferix Remote PC

In the future, Inferix plans to develop Inferix Remote PC, a service that allows end-users to directly access Workers via remote desktop. This will enable users to have greater control over installing the necessary software on the worker machines, potentially reducing service costs. For example, if a user has a license for a render engine that Inferix's workers do not have, they can still utilize it with Inferix's computational hardware through Inferix Remote PC.

D. Rendering Professional Network

In its plan to utilize the Community Grants Fund, Inferix has programs aimed at developing a community of graphic artists and students from universities related to visual computing. These initiatives will result in the creation of a Rendering Professional Network capable of connecting end-users with professional graphic creators. This will foster sustainable growth for Inferix.

References

[1] J. F. Hughes, A. van Dam, M. McGuire, D. F. Sklar, J. D. Foley, S. K. Feiner, and K. Akeley, *Computer Graphics: Principles and Practice*, 3rd ed. Addison-Wesley, 2014.

[2] M. Wu and B. Liu, "Watermarking for image authentication," in *Proceedings of International Conference on Image Processing*, 1998.

[3] M. M. Yeung and F. Mintzer, "An invisible watermarking technique for image verification," in *Proceedings of International Conference on Image Processing*, 1997.

[4] R. B. Wolfgang and E. J. Delp, "Fragile watermarking using the vw2d watermark," in *Security and Watermarking of Multimedia Contents*. SPIE, 1999.

[5] C. Gentry, "Fully homomorphic encryption using ideal lattices," *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, 2009.

[6] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. P. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, 2022.

[7] I. J. Cox, J. Kilian, T. Leighton, and T. Shamoon, "Secure spread spectrum watermarking for multimedia," *IEEE Transactions on Image Processing*, 1997.

[8] I. J. Cox, M. L. Miller, and A. L. McKellips, "Watermarking as communications with side information," *Proceedings of the IEEE*, 1999.

[9] S. A. Craver, N. D. Memon, B.-L. Yeo, and M. M. Yeung, "Can invisible watermarks resolve rightful ownerships?" in *Storage and Retrieval for Image and Video Databases*, 1997.

[10] N. Jayant, J. Johnston, and R. Safranek, "Signal compression based on models of human perception," *Proceedings of the IEEE*, 1993.

[11] H. R. Wu, A. R. Reibman, W. Lin, F. Pereira, and S. S. Hemami, "Perceptual visual signal compression and transmission," *Proceedings of the IEEE*, 2013.

[12] Q. Lian, Z. Zhang, M. Yang, B. Y. Zhao, Y. Dai, and X. Li, "An empirical study of collusion behavior in the maze p2p file-sharing system," in *27th International Conference on Distributed Computing Systems*, 2007.

[13] I. J. Cox, M. L. Miller, , and A. L. McKellips, "Watermarking as communications with side information," *Proceedings of the IEEE*, 1999.

[14] C. Cachin, "An information-theoretic model for steganography," in *International Workshop on Information Hiding*, 1998.

[15] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn, "Information hiding - a survey," *Proceedings of the IEEE*, vol. 87, no. 7, 1999.

[16] M. D. Swanson, B. Zhu, and A. H. Tewfik, "Transparent robust image watermarking," in *Proceedings of 3rd IEEE International Conference on Image Processing*, 1996.

[17] C. I. Podilchuk and W. Zeng, "Image-adaptive watermarking using visual models," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 4, 1998.

[18] S. Voloshynovskiy, A. Herrigel, N. Baumgaertner, and T. Pun, "A stochastic approach to content adaptive digital image watermarking," in *International Workshop on Information Hiding*, 2000.

[19] J. H. Reif, D. J. Tygar, and A. Yoshida, "Computability and complexity of ray tracing," *Discrete and Computational Geometry*, vol. 11, no. 3, 1994.

[20] J. Stern, "Secret linear congruential generators are not cryptographically secure," in *28th Annual Symposium on Foundations of Computer Science*, 1987.

[21] J. Chou, S. S. Pradhan, and K. Ramchandran, "On the duality between distributed source coding and data hiding," in *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers*, vol. 2, 1999.

[22] Actif3D Team. (2023) The no-code vr/ar platform for creator economy. [Online]. Available: https://academy.actif3d.com/blog:no-code-vr-platform

[23] TensorOpera. (2023, Nov) Scalellm: Unlocking llama2-13b llm inference on consumer gpu rtx 4090, powered by fedml nexus ai. [Online]. Available: https://blog.tensoropera.ai/scalellm-unlocking-llama2-13b-llm-inference-on-consumer-gpu-rtx-4090-powered-by-fedml-nexus-ai/

[24] ——. (2022) Scalellm: Serverless and memory-efficient model serving engine for large language models. [Online]. Available: https://docs.tensoropera.ai/deploy/scalellm

[25] Global Market Insight. (2024, May) Visual computing market size - by component (hardware, software), by display platform (interactive whiteboards, interactive kiosk, interactive table, interactive video wall, monitors), by industry verticals, regional outlook and forecast, 2024 - 2032. [Online]. Available: https://www.gminsights.com/industry-analysis/visual-computing-market

[26] Polaris Market Research. (2022, Sep) Federated learning market share, size, trends, industry analysis report, by application (industrial internet of things, drug discovery, risk management, augmented and virtual reality, data privacy management, others); by industry vertical; by region; segment forecast, 2022 - 2030. [Online]. Available: https://www.polarismarketresearch.com/industry-analysis/federated-learning-market
[27] A. Haleem, A. Allen, A. Thompson, M. Nijdam, and R. Garg, "Helium: A decentralized wireless network," Helium Systems, Tech. Rep., 2018.
[28] P. Snow, B. Deery, J. Lu, D. Johnston, and P. Kirby, "Factom: Business processes secured by immutable audit trails on the blockchain," Factom Protocol, Tech. Rep., 2018.
[29] K. Samani. (2018, Feb) New models for utility tokens. [Online]. Available: https://multicoin.capital/2018/02/13/new-models-utility-tokens/
[30] Super Renders Farm. (2024) Render farm pricing. [Online]. Available: https://superrendersfarm.com/pricing
[31] RebusFarm. (2024) Render farm prices and discounts. [Online]. Available: https://rebusfarm.net/buy/products
[32] Fox Renderfarm. (2024) Render farm pricing. [Online]. Available: https://www.foxrenderfarm.com/pricing.html

Appendix A
Proofs

A. Fourier transform of complex atomic signals

Proof. Substitute the atomic signal eq. (10):

$$w_i(x,y) = Ae^{2i\pi\left(\frac{x}{X_i}+\frac{y}{Y_i}\right)} \quad (0 \le x < M, 0 \le y < N)$$

into the discrete Fourier transform:

$$F(u,v) = \frac{1}{M\times N}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} w_i(x,y)e^{2i\pi\left(\frac{xu}{M}+\frac{yv}{N}\right)}$$

We have

$$F(u,v) = \frac{A}{M\times N}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} e^{2i\pi\left(\frac{x}{X_i}+\frac{y}{Y_i}\right)}e^{2i\pi\left(\frac{xu}{M}+\frac{yv}{N}\right)}$$

Moreover

$$\begin{aligned}
&\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} e^{2i\pi\left(\frac{x}{X_i}+\frac{y}{Y_i}\right)}e^{2i\pi\left(\frac{xu}{M}+\frac{yv}{N}\right)}\\
=&\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} e^{2i\pi x\left(\frac{1}{X_i}+\frac{u}{M}\right)}e^{2i\pi y\left(\frac{1}{Y_i}+\frac{v}{N}\right)}\\
=&\left(\sum_{x=0}^{M-1} e^{2i\pi x\left(\frac{1}{X_i}+\frac{u}{M}\right)}\right)\left(\sum_{x=0}^{N-1} e^{2i\pi x\left(\frac{1}{Y_i}+\frac{v}{N}\right)}\right)
\end{aligned}$$

Using geometric sum formulae:

$$\begin{aligned}
=&\frac{1-e^{2i\pi M\left(\frac{1}{X_i}+\frac{u}{M}\right)}}{1-e^{2i\pi\left(\frac{1}{X_i}+\frac{u}{M}\right)}}\frac{1-e^{2i\pi N\left(\frac{1}{Y_i}+\frac{v}{N}\right)}}{1-e^{2i\pi\left(\frac{1}{Y_i}+\frac{v}{N}\right)}}\\
=&\frac{1-e^{2i\pi\frac{M}{X_i}+2i\pi u}}{1-e^{2i\pi\left(\frac{1}{X_i}+\frac{u}{M}\right)}}\frac{1-e^{2i\pi\frac{N}{Y_i}+2i\pi v}}{1-e^{2i\pi\left(\frac{1}{Y_i}+\frac{v}{N}\right)}}\\
=&\frac{1-e^{2i\pi\frac{M}{X_i}}}{1-e^{2i\pi\left(\frac{1}{X_i}+\frac{u}{M}\right)}}\frac{1-e^{2i\pi\frac{N}{Y_i}}}{1-e^{2i\pi\left(\frac{1}{Y_i}+\frac{v}{N}\right)}}
\end{aligned}$$

Hence:

$$F(u,v) = \frac{A}{M\times N}\frac{1-e^{2i\pi\frac{M}{X_i}}}{1-e^{2i\pi\left(\frac{1}{X_i}+\frac{u}{M}\right)}}\frac{1-e^{2i\pi\frac{N}{Y_i}}}{1-e^{2i\pi\left(\frac{1}{Y_i}+\frac{v}{N}\right)}}$$

□

B. Convergence of energies

Proof. The random variables $\mathcal{X}_i, \mathcal{Y}_i\ (i \in \mathbb{N})$ are independent then two variables:

$$\overline{\mathcal{X}}_n = \frac{\mathcal{X}_1 + \ldots \mathcal{X}_n}{n} \quad \text{and} \quad \overline{\mathcal{Y}}_n = \frac{\mathcal{Y}_1 + \ldots \mathcal{Y}_n}{n}$$

are independent for all $n \in \mathbb{N}$. Since $\mathcal{X}_i\ (i \in \mathbb{N})$ are independent and identically distributed and $\mathcal{X}_i \sim \mathcal{N}\left(\mu, \sigma^2\right)$

$$\overline{\mathcal{X}}_n \xrightarrow[n\to\infty]{a.s} \mu$$

by strong law of large numbers; similarly $\overline{\mathcal{Y}}_n \xrightarrow[n\to\infty]{a.s} \mu$. Hence

$${}^t\begin{bmatrix}\overline{\mathcal{X}}_n & \overline{\mathcal{Y}}_n\end{bmatrix} \xrightarrow[n\to\infty]{a.s} {}^t\begin{bmatrix}\overline{\mathcal{X}} & \overline{\mathcal{Y}}\end{bmatrix}$$

for some independent and identically distributed variables $\overline{\mathcal{X}} \sim \overline{\mathcal{Y}} \sim \mu$. Let us fix some $0 \le u < M$ and $0 \le v < N$, then it is obvious that the function

$$F_{(u,v)} \triangleq (x,y) \mapsto \frac{A}{M\times N}\frac{\left(1-e^{2i\pi\frac{M}{x}}\right)\left(1-e^{2i\pi\frac{N}{y}}\right)}{\left(1-e^{2i\pi\left(\frac{1}{x}+\frac{u}{M}\right)}\right)\left(1-e^{2i\pi\left(\frac{1}{y}+\frac{v}{N}\right)}\right)}$$

is continuous. We define the variable:

$$\begin{aligned}
\mathcal{F}_{(u,v)}: \Omega &\to \mathbb{R}^2\\
\omega &\mapsto F_{(u,v)}\left(\mathcal{X}(\omega), \mathcal{Y}(\omega)\right)
\end{aligned}$$

for some random variables $\mathcal{X}$ and $\mathcal{Y}$, then $\mathcal{F}_{(u,v)}(\omega)$ is nothing but the Fourier transform of the atomic signal whose the horizontal and the vertical period are $\mathcal{X}(\omega)$ and $\mathcal{Y}(\omega)$ respectively. Let us consider the sequence $\left(\mathcal{F}^n_{(u,v)}\right)_{n\in\mathbb{N}}$ where:

$$\mathcal{F}^n_{(u,v)} \triangleq \omega \mapsto F_{(u,v)}\left(\overline{\mathcal{X}}_n(\omega), \overline{\mathcal{Y}}_n(\omega)\right)$$

By the continuous mapping theorem:

$$\mathcal{F}^n_{(u,v)} \xrightarrow[n\to\infty]{a.s} \omega \mapsto F_{(u,v)}\left(\overline{\mathcal{X}}(\omega), \overline{\mathcal{Y}}(\omega)\right)$$

Since we have proved that $\overline{\mathcal{X}} \sim \overline{\mathcal{Y}} \sim \mu$ then the limit of the sequence is the constant variable $F_{(u,v)}(\mu,\mu)$. Also, since $F_{(u,v)}$ is continuous:

$$\left|F_{(u,v)}\left(\overline{\mathcal{X}}_n, \overline{\mathcal{Y}}_n\right) - \overline{F}_{(u,v)}\left(\mathcal{X}_n, \mathcal{Y}_n\right)\right| \xrightarrow[n\to\infty]{} 0$$

Hence $\overline{F}_{(u,v)}\left(\mathcal{X}_n, \mathcal{Y}_n\right) \xrightarrow[n\to\infty]{a.s} F_{(u,v)}(\mu,\mu)$. □

Remark. *The proof does not require that X_i and Y_i have the same distribution. Indeed, if $X_i \sim \mathcal{N}(\mu_x, \cdot)$ and $Y_i \sim \mathcal{N}(\mu_y, \cdot)$ then $\overline{F}_{(u,v)}\left(\mathcal{X}_n, \mathcal{Y}_n\right) \xrightarrow[n\to\infty]{a.s} F_{(u,v)}\left(\mu_x, \mu_y\right)$.*

Appendix B
Price simulation details

The table in fig. 17 contains price simulation detailed data. We are calculating the Provider's input costs, which include electricity costs and hardware depreciation. The calculations are standardized for a single GPU device.

- *Electricity Cost* is calculated based on the power consumption of the device multiplied by the assumed unit price of $0.20 per kWh. Hardware depreciation costs

GPU Model	Power Consumption	Electricity Cost	GPU Hardware Price	GPU Depreciation Cost	Other Hardware Price	Other HW Depreciation Cost	Provider Total Cost (1)	Provider Profit/ Hour(2)	Provider Price (1+2)	Provider Payback time (months)	Inferix Service Price
RTX3060	170	$0.034	$314	$0.009	$250	$0.007	**$0.050**	**$0.050**	**$0.100**	**16**	**$0.188**
RTX3070	220	$0.044	$450	$0,013	$350	$0.010	**$0.067**	**$0.067**	**$0.134**	**17**	**$0.227**
RTX3080	320	$0.064	$750	$0,021	$750	$0.022	**$0.107**	**$0.107**	**$0.214**	**19**	**$0.323**
RTX3090	355	$0.072	$1500	$0,043	$850	$0.025	**$0.138**	**$0.138**	**$0.277**	**24**	**$0.398**
RTX4090	450	$0.090	$2000	$0,057	$1000	$0.029	**$0.176**	**$0.176**	**$0.352**	**24**	**$0.488**

Figure 17: Price simulation details

are calculated based on a general assumed depreciation rate of 25% per year for both the GPU and other PC components.
- *Provider Price:* The suggested unit price for Providers is calculated as double the Provider's input costs per GPU per hour.
- The input cost for Inferix is calculated by adding the storage costs to the Provider price. This storage cost is based on the assumption that each 3D scene requires 1 GB of short-term storage for 96 hours and 100 MB of long-term storage, with a rate of $0.05/GB per month.
- *Inferix Service Price* is calculated by adding a 20% commission for Inferix Foundation to the input cost.

APPENDIX C
HARDWARE REQUIREMENTS FOR NODES

A. Manager Node

The minimum requirements for a single license for Manager Node are as follows:

- 6 x64 CPU Core@ 2.1 GHz
- 64 GB RAM
- 1000 GB disk space
- 100 Mps internet connection

B. Standard Verifier Node

The minimum requirements for a single license for Standard Verifier Node are as follows:

- 1 x64 CPU Core@ 2.1 GHz
- 8 GB RAM
- 10 GB disk space
- 10 Mps internet connection

C. Standard Unit Node

The hardware requirements for a Standard Unit Node must be ***exactly*** as follows:

- 1x Intel®Core™i9 CPU
- 32 GB RAM
- 20 GB SSD disk space
- 100 Mps internet connection

D. Mobile Verifier Node

The minimum requirements for a single license for Mobile Verifier Node are as follows:

- Octa-core (2.2 GHz Cortex-A55 or equivalent)
- 32 GB RAM
- 10 GB disk space
- 10 Mps internet connection

Remark. *Mobile Verifier Node can be used only for PoR verification tasks. This type of node cannot run other types of verification.*

APPENDIX D
PERFORMANCE EVALUATION

The detailed data for the performance evaluation is given in tables III to VI hereafter. In any table, each row shows the execution time (in second) of inserting a vector noise into a Blender scene whose name is given in the table name, and the time (also in second) of verifying the distortions regions on a frame rendered from the scene.

The tests are proceeded on a workstation of Intel®Core™i5 2.5 GHz CPU, 32 GB RAM. The noise insertion and noise verification do not need GPU.

Vector length	Insertion time	Verification time
2	0.747	0.343
5	0.785	0.845
7	0.822	1.059
10	0.871	1.525
12	0.923	1.855
15	0.982	2.352
17	1.061	2.612
19	1.062	2.929
20	1.092	3.081
21	1.094	3.217
23	1.163	3.508
25	1.204	3.786
27	1.241	4.182
29	1.330	4.475
30	1.335	4.673
32	1.394	5.059
35	1.472	5.585
37	1.490	5.982
40	1.612	6.409

Table III: Coca-Cola scheme

Vector length	Insertion time	Verification time
2	0.920	2.047
5	1.300	1.077
7	1.524	1.412
10	2.085	1.976
12	2.239	2.286
15	2.550	2.722
17	2.865	3.068
19	3.153	3.453
20	3.269	3.527
21	3.439	3.747
23	3.592	4.123
25	3.975	4.546
27	4.279	4.913
29	4.594	5.313
30	4.666	5.557
32	5.087	5.812
35	5.434	6.265
37	5.729	6.654
40	6.229	7.116

Table IV: Grease Pencil Bike

Vector length	Insertion time	Verification time
2	3.959	0.546
5	7.125	0.922
7	7.468	1.251
10	8.938	1.642
12	9.934	1.871
15	12.773	2.443
17	13.721	2.751
19	14.028	3.083
20	15.351	3.228
21	15.766	3.347
23	17.384	3.717
25	18.587	3.978
27	21.182	4.281
29	21.549	4.523
30	22.224	4.706
32	23.549	5.037
35	27.202	5.444
37	29.063	5.764
40	29.277	6.087

Table V: Blender 3.5 Splash

Vector length	Insertion time	Verification time
2	2.313	0.657
5	5.012	1.152
7	10.228	1.498
10	15.495	1.997
12	17.667	2.415
15	19.627	2.964
17	23.986	3.311
19	26.091	3.738
20	27.808	3.816
21	28.326	4.080
23	31.014	4.479
25	32.523	4.770
27	34.605	5.151
29	36.062	5.462
30	36.456	5.710
32	38.921	6.031
35	42.441	6.496
37	44.138	6.919
40	49.456	7.458

Table VI: Bathroom Above Corner